Integrated Electronic Health Record Access by Object Versioning and Metadata

Tore Mallaug[1,2], Kjell Bratbergsengen[2]

[1] Faculty of Informatics and e-Learning,
Sør-Trøndelag University College
NO-7004 Trondheim, Norway
torem@aitel.hist.no
[2] Department of Computer and Information Science,
Norwegian University of Science and Technology
NO-7491 Trondheim, Norway
{torem, kjellb}@idi.ntnu.no

Abstract. Digitization of personal health data facilitates easier and timely data access in integrated healthcare services and in medical or (historical) demographic research. We suggest a common middle layer framework that represents data content, related schemas and possible ontologies as temporal object versions on a timeline. Bidirectional and bi-temporal object versioning are included. The versioning is extended to accept mappings to none-existing (potential) data value versions in time. We enrich this framework by including temporal representation of mappings, and by adding metadata elements related to temporal data change and mapping generation. Such metadata is useful processing database requests from different local applications in the time space. The paper also provides examples and a short discussion of certain useful metadata elements. The usage is exemplified by XML with elements related to the Electronic Health Record (EHR) case.

1 Introduction

1.1 A Data Versioning Solution for the EHR-case

In a modern integrated health care system the need for collected personal health data is increasing [1,2,3]. To establish an independent national personal EHR (Electronic Health Record) [4] centered around each person makes it easier for citizens to collect, manage and control their own health data. This future EHR stores both the most updated version and all older versions of a person's health data. The data is collected from different service providers (hospitals, doctors, dentists, ...), different health related registers and other information on general health conditions, living conditions, and more over time. This scenario asks for a safe and timely common access to the overall EHR system, managed under tight restricted access control, Today such a service does not exist. Message passing exchange between heterogeneous fragmented health information systems is the only way to aggregate health data about a given person. Our approach is to build a common (national) middle layer framework for such a data access and data collection, linked to a common database solution for the integrated EHR. We are going from a message passing system to a data sharing system. Because of legal problems and organizational problems a realization of our middle layer and a related database is impossible for the time being. A fully independent national personal health record database requires reforms in laws. The management of such a national database must be independent of service providers, and service providers must be by law obliged to document their actions in this database.

The middle layer framework uses a temporal object data model (shortly described in [6]) that allows object versioning (like schema versioning [7]) to make it possible for local applications to read (access) and write (store) personal health data on different heterogeneous formats, for example formats related to aging terminology or aging ontology. The object versioning can be implemented in a common middle layer system. We choose to call this system the *DRL* – short for the *Data Representation Layer*.

We are also examining how different metadata about a mapping between object versions can help in interpretation of data. Such metadata can include information about why the mapping is needed (the cause of data changes, in some cases this is trigged by temporal changes) and how the mapping is (was) generated, including information about the mapping (match) type and the chosen mapping approach in the mapping generating process. Note that all mapping examples in the paper are sub-mappings between sub-elements in two object versions.

1.2 Research Objectives

In this paper we focus on some versioning problems for our temporal object data model on the middle layer. The database layer is not discussed. Different research questions can be discussed concerning an implementation of our DRL (Data Representation Layer) system. We touch three questions in the following:

1. *How to represent mappings between object versions?* Each individual (sub-) mapping has to be related to a mapping rule represented in a well-known language, for example XSLT [8].
2. *How to approximate inexact (sub-) mappings when needed and how to handle impossible (sub-) mappings?* To this question we include how to represent metadata about approximations, and how to representation errors if a requisite result is impossible to generate. An approximation may be motivated by medical profession reasons related to the semantic of the data.
3. *How to represent and store metadata about causes of data changes*. The causes are linked to temporal changes, both structural and semantically changes [6]. In some cases data changes are related to changes in ontologies [9], for example changes in the conceptualization of the data.

1.3 Temporal Object Data Model

It is necessary to represent and store different temporal objects (instances) and relationships between objects in the time space. We need object versions with their own global temporal OID, a way to represent and store sets of different time stamps related to each object (the model is bi-temporal [10,11]), and a way to represent and encapsulate the stored data content in object versions.
We use object versioning to handle different requests (database queries) from local user applications, it means we handle temporal versioning both forward and backward in time. We choose to implement this by mappings between object versions, and to represent and store the mappings as own temporal objects as well. We do not present a single mapping algorithm for generating new mappings between object versions. Our suggested data model for the framework has to be general enough to accept such temporal changes in data representation over a very long-term time span, like a person's lifetime (100 years).

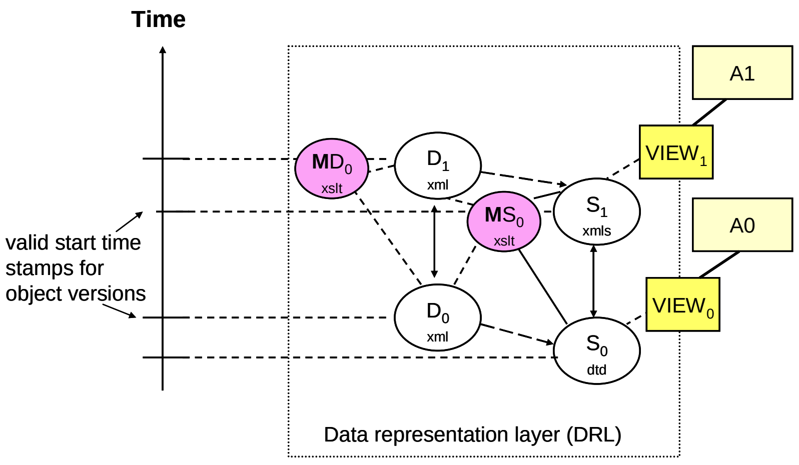
Time
A1
VIEW₁
A0
VIEW₀
MD₀ xslt
D₁ xml
MS₀ xslt
S₁ xmls
D₀ xml
S₀ dtd
valid start time stamps for object versions
Data representation layer (DRL)

Fig. 1. Example of temporal object versions in the DRL

Fig.1 illustrates some object versions and links between them in the DRL. Each object version is temporal and has an own start time stamp (start time may indicates the transaction-time [10,11] for an object) on a time axis in the DRL-system. In Fig.1 we operate with three different object types. *Data objects* (D_0 and D_1) are validated according to schemas stored in own *schema objects* (S_0 and $S_{1,}$ respectively). A bidirectional schema mapping between S_0 and S_1 is represented by an own *mapping object* denoted MS_0. In addition it is possible to have a mapping object denoted MD_0 for representing a bidirectional data value mapping between D_0 and D_1, that we call data (content) versioning (instance-level in [12]). In some cases we may have data versioning between instances of the same schema version - means versioning without any schema update. Also, data versioning can be needed if a schema versioning (mapping) gives inexact results for some of the sub-elements involved.

Fig. 1 also shows two local applications denoted A0 and A1. A0 and A1 are implemented to access and store data of the type D as given by S_0 through $VIEW_0$ and by S_1 through $VIEW_1$, respectively. A0 may needs to access D_1 through S_0, while A1 may needs to access D_0 through S_1. We do not require A0 to be re-implemented or recompiled to read data in D_1 [7] or to write new data versions in the most updated format (here S_1). If both D_0 and D_1 exist in the DRL, a direct value (data) mapping between sub-elements can be generated and stored in MD_0. However, if D_1 does not exist yet, potential (new) values according to S_1 for sub-elements in D_0 may be needed (Section 3.3.1 shows a similar situation backward in time). It is less likely that old EHR-data is updated if e.g. medical practice is changing and a new schema version is introduced, and new EHR-data may be converted to old formats only when required (as in lazy mechanism of converting data [13]).

In the following, section 2 shortly mentions how our research relates to other work. Section 3 describes how different metadata elements can be used related to different temporal issues and mapping problems. Section 4 shortly shows an example using XSLT. Section 5 summarized.

2 Related Works

2.1 Data and Schema Versioning

In our data model (in DRL) we adopt the following definitions from schema versioning [7]: A database system supports *schema evolution* if it permits modification of the database schema without loss of data. A database accommodates *schema version* if it allows the querying of all data, both retrospectively and prospectively through definable version interfaces ($VIEW_0$ and $VIEW_1$ in Fig.1). Our approach allows both schema and data (instance) versioning, as [12] distinguished between schema- and instance-level. This is implemented by a log-only solution (e.g. [14]), where no old schema or data content versions are ever deleted, but any update is stored as a new object version. DRL has to represent the versioning process, and the implementation for the bidirectional mapping between versions in time. This means that DRL has to represent a schema mapping generating process between already defined (ready to use, and in some cases already in use) schemas that are not necessary meant for being a part of such a versioning.

2.2 Integrated Health Information Systems

It is a lack of interoperable EHR systems (e.g. [15]). Studies on an integrated EHR [3,16,17,18] so far suggest message exchange solutions through a middle layer in health data networks or Internet. None of these projects focus on common temporal data representation frameworks or common database solutions.

Implemented standards and solutions are related to electronic massage passing exchange between local systems, for example MedXML [19] and HL7 [20]. Such standards typically represent health data as single documents in the time space. It is possible to link documents in MedXML, but the standard does not give any hints about versioning between new and old data elements. The MedXML documentation assumes that the most updated data is the data of interest though this is not a limitation in MedXML's message standard. Mapping solutions are suggesting mapping to and from a common message passing format, but not between different local representations.
It is research on representation of change in medical terminologies, e.g. [21], and metadata in interoperability between health information systems is discussed e.g. by [22]. Our use of metadata is not directly related to medical usage, but our general framework approach in DRL.

3 Metadata Elements for Data Change and Mapping

3.1 Object Versioning

Metadata is data that describes other data to enhance its usefulness. In our case we add metadata elements to the representation of (sub-) mappings in object versioning. In this section the meaning and usage of these elements are discussed. Section 4 summarized some of these elements in a XSLT-template. We can call a (metadata) element for representing a sub-mapping itself: **`subMapping`** - a complex element representing a sub-mapping between elements in two objects.

3.2 Metadata about Causes of Data Change

We include the metadata element: **`cause`** - this element shows the reason for a given data change, and by that the reason for a mapping between two versions of the data. The value of the element can be one (or many) standardized constant, that indicates for example (possible causes discussed in [6]) a simple entity update (no structural or semantically change), a temporal change without any structural or semantically change, or a temporal change caused by structural and / or semantically schema or ontology change, e.g. cases where ontology evolution is applicable [9]. An extension of the model in Fig.1 is to represent a temporal change as an own object with a temporal OID as well, since the change has an influence on the internal query and mapping processing (see the `temporalHandlers` element) in the DRL.

3.3 Temporal Data Change Perspectives

The object versioning has to be seen in relation to different temporal perspectives, or dimensions, on a time axis. It is important to call attention to the fact that our versioning not only focusing on the newest, or most updated, versions of data content, but has to be able to handle requests from applications that need older data versions and older data representations (schema and ontology versions) as well. Such a perspective gives several temporal problems that can be discussed.
A possible metadata element: **`temporalHandlers`** - a set of sub-elements concerning the different temporal change perspectives. Such metadata can for example be useful for tools in the DRL when processing queries, e.g. to consider constraints on results from a specific time interval. Below we shortly discuss some examples related to such temporal perspectives.

3.3.1 "Overlapping" Temporal Changes

We say that two or more temporal changes are "overlapping" if (sub-) changes caused by one of these temporal changes may results in "loss" of semantically knowledge about specific data content updates. To exemplify the problem we use a simple case showing updates on citizen's postal address code. This particular case is partly spatial, and can be used in demographically health research. The postal code 7873 in Fig.2 is not in use anymore on the date 01.10.2005 and citizens having the code 7873 are split into two possible new codes, 7800 NO or 7700 NO (This change in the Norwegian postal address system is hypothetical for the matter of the example). In this example we have a value mapping from $D_0 \rightarrow D_1$, that is ('7873' → '7800 NO') or ('7873' → '7700 NO'). For object instances (citizens) that exist before this temporal change at the date 01.10.2005, a semi-automatically tool in DRL can choose which of the two alternative mappings above that work for every specific object (in co-operation with a system end user). At the same time as the mapping generating process decides the right mapping for $D_0 \rightarrow D_1$, the value mapping $D_1 \rightarrow D_0$ is also generated to make the mapping bidirectional.

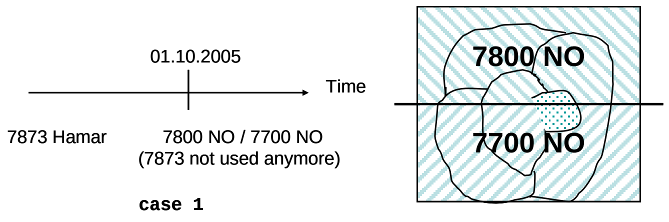

Fig. 2. Example of temporal update of a postal code (from [6])

However, if a new citizen moves to the geographically post block '7800 NO' after the change (after 01.10.2005), we do not know if the value mapping ('7800 NO' →

‘7873’) holds for this new instance. From an ordinary object versioning point of view such a mapping back in time is not meaningful since this person did not live at ‘7873’ before the actual temporal change, but for our bidirectional view this problem is relevant. Say if the local application A0 requests the postal code in the (old) format of ‘7873’ after the change of 01.10.2005 (as in Fig.1), we can only return citizens that lived at ‘7873’ at the time before the change, not the new citizens that lives there after the change. This example shows that an “old” temporal change can give “loss” of semantically knowledge in future data changes (in this case a future entity update). This means that a temporal change not only influence existing object versions by the time the change occurs, but may also influence future object versions that are introduced after the temporal change in time.
This problem asks for an own separate representation of the temporal change as a temporal object with an OID. Such a temporal change representation makes it possible to generate mappings back in time when a new object (instance) is introduced in the time space, since the DRL tool then can search back to find temporal changes of interest during the process of generating bidirectional mappings for the new object version.

3.3.2 Indirect Mappings

Say, if we have an object version A, and a newer introduced version B, there might be a sub-mapping from A to B that is impossible or inexact. Later in time a newer version C is introduced, and the sub-mapping from A to C can still be possible even if A $\rightarrow$ B does not work. In real this situation asks for a change to generate a “new” sub-mapping from A by the time C is introduced, or in other words a “re-generating” of the mappings from A forward in the time space.
An example is if the postal code 7873 belonging to the post office “Hamar” is not in use by the postal service for a given time period and citizens having code 7873 are given the neighbor post office code 7870 “Oslo” from 01.10.2000 to 01.10.2002. At 01.10.2002 the code 7873 is reopened (Fig. 3).

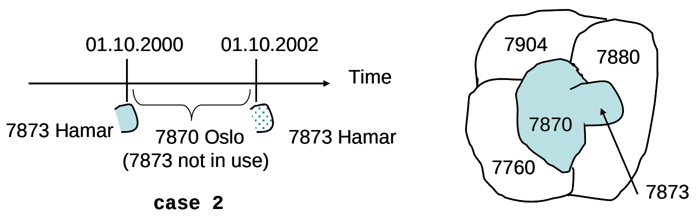

Fig. 3. Example of temporal update of a postal code (from [6])

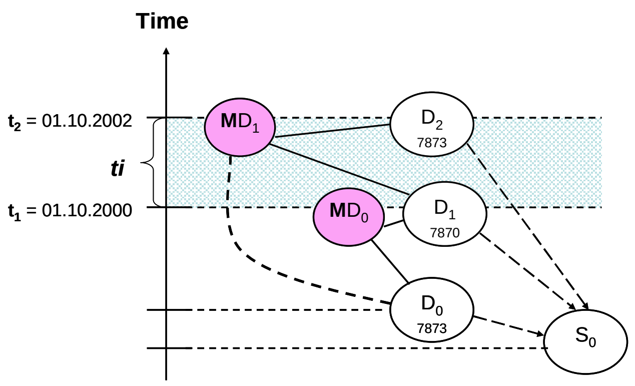

Fig. 4. Example of object versions and a time interval `ti`

If a person moves to “Hamar” during the time sequence between 01.10.2000 and 01.10.2002, it is impossible to generate a mapping back in time to the period before 01.10.2000 since “Hamar” does not exists as a post office at the time this person moves to the area. However, after 01.10.2002 the person’s postal code is updated from 7870 to 7873, and by then a mapping back to before 01.10.2000 is possible (it is an equal value map 7873 → 7873, a link from D_0 to MD_1 is illustrated as a dotted line in Fig. 4).

3.3.3 “Impossible” Temporal Database Queries

For a certain time interval a given query can be “impossible” to process, or returns an inaccurate result. Say we have a query Q1 and a time interval given by the time stamps t_1 and t_2. Processing Q1 before t_1 and after t_2 can be ok, but in between t_1 and t_2 (illustrated as the time interval `ti` in Fig. 4) on the time axis Q1 may give a wrong or inexact result. This problem is from temporal databases where a query returns different results when evaluated at different times [11].
If t_1 and t_2 are the dates in postal code example in Fig. 3 and if Q1 is the following SQL SELECT:

```
SELECT Person.* FROM Person
WHERE post_office = ‘Hamar’ AND date = ‘01.01.2001’;
```

At the date Q1 requires, the post office value is “Oslo” instead of “Hamar”. The office element was updated at the date 01.10.2000 when “Hamar” stopped to exist as a post office, and the citizens in “Hamar” got the same post office as for the citizens of “Oslo”. However, after that update it is impossible to know who are living in “Hamar”, and because of that it is difficult to process Q1 for the required date. To return all citizens of “Oslo” as a result (meaning using a value mapping from “Hamar” to “Oslo”, that is a legal mapping from an object versioning point of view) will return too many entities (citizens).

3.4 Mapping Approach and Problematic Mappings

In general a mapping that works in both direction has to be 1:1 (one-to-one and injective). A mapping that is M:1 (many-to-one) in one direction does not have any inverse mapping. For example a well-defined functional approach on implementation of mapping rules is then impossible. In such cases a 1:1 -mapping has to be generated by adding knowledge about the semantic of the data directly from medical professional end-users or if possible by using knowledge given by ontologies, or the mapping is terminated (if such a solution is accepted). Such a case can for example be caused by changes in the data value domain of an element. An example is if the blood pressure is represented as an integer value in one schema version and as a string value (e.g. a value domain of the possible values; "low", "normal", or "high") in another version. For example a "high" blood pressure is a value higher than 130 +/-2 (mm Hg systolic pressure), meaning we have critical values from 128 to 132, and the mapping result (in the direction from the string to the integer) is never exact. Such a problem is critical if the local application acceptance of approximation in the returned result is low or not accepted.
Recall Fig.1. Say D_0 stores the blood pressure value as the string above, while D_1 stores the same value as the integer above. We may have the following scenario: The most updated version of the blood pressure element has the value "high" and is stored in D_0. A local application A1 wants to read the value as an integer type according to a schema version S_1, but D_1 does not exist - it is just a potential version. An approximation has to be done when generating a value mapping D_0.string → D_1.integer. The reason for the approximation can be based (and justified) by a medical domain specific approach related to the semantic of the data. The approach can also be given by a temporal conceptual change in the interpretation of the result, like a paradigm shift in medicine. In this example two general mapping solutions are as follow:
Mapping: D_0.string → D_1.integer

`Solution 1`: "high" → *n*, where *n* is an integer digit validated by S_1
`Solution 2`: "high" → TERMINATED (an impossible mapping)

First, the fact that the mapping result is generated by a specific approach, like a number approximation, is stored and linked to information about the mapping algorithm (represented by syntax readable for applications in the future). Second, the *reason* why the approach was chosen in this particular case is stored as well. Some comments on each solution:
`Solution 1`: The decision of a selected approach is taken by a health care professional end-user based on a medical assessment. Such assessments can be based on known (medical) ontologies (e.g. ontologies mentioned in [23]), or by present practice in health care or medicine.
`Solution 2`: In the "terminated" case we suggest that the DRL offers a set of well-defined constants used to describe "default" reasons of why a number is not present in D_1. This can still be interpreted as a legal value, even if no number value is given for D_1. However, it is not sure that A1 can read such a constant, so this solution does not work in all cases, e.g. if the value is critical in emergency. Possible constants are

(but not limited to): NULL that indicates an empty value (as in databases), ERROR that indicates an mapping attempt that was considered as impossible during the mapping generation process (given that such an error is acceptable in the particular case), and TERMINATED that indicates an end of a temporal line of (sub-) element versions in the temporal space (for example caused by a paradigm shift in medicine).
The same solutions for the above example can be discussed for the opposite temporal direction: Mapping: D_1.integer → D_0.string

> `Solution 1`: 130 → *s*, where *s* is a string validated by S_0
> `Solution 2`: TERMINATED → ERROR

`Solution 1`: An approach can decide that 130 in D_1 are transformed to "high" in D_0. However, this mapping works for any value in the range from 128 and higher in D_1, so a question is if this approximation is acceptable. Using this solution the process is equal to the mapping $D_0 \rightarrow D_1$. A professional end user has to make a decision on how the value 130 (if this value in D_1) is transformed to a legal string in D_0.
`Solution 2`: In the "terminated" case it is impossible to generate a value for D_0 back in time, unless an end-user can generate the value mapping manually.
A metadata element: **`mappingInfo`** - shows information about a mapping and the related mapping generating process, like the mapping approach (or matching) related to different types in a classification of schema matching [12], e.g. a match can be linguistic (name) based or constraint-based. Another mapping approach is to build mappings on a generic schema evaluation across data models as in [24].

3.5 Audit Data

A metadata element: **`audit`** - shows audit data about the mapping process, and has a logging purpose. What were done by whom? Audit data is not discussed in this paper.

4 An Implementation of a Mapping and Metadata

Presume that the data content of the different objects in Fig. 4 (from the postal code example in Fig. 3) is represented in XML-technology. The following XSLT-template is a part of the data content of the mapping object MD_0 and shows metadata elements for the mapping from D_0 to D_1 at the time stamp t_1. A similar metadata set can include the mapping from D_1 to D_0 as well. The element `subMapping` represents the mapping between the two postal codes. The complex element `metadata` includes the sub-elements mentioned in Section 3.2 to 3.4. A hierarchical structure of sub-elements can represent information of different levels of details. More research is needed on `temporalHandlers` to support help in the temporal problems of Section 3.3. The element `tempQueryHandle` can be used to identify change in the "post_office" entity (specified by S_0) and to identify interval *ti* when generating metadata for MD_1 at t_2.

```
<xsl:template match="/">
<xsl:element name="subMapping">
 <xsl:element name="postcode">
  <xsl:value-of select="element/postcode"/>
 </xsl:element>

 <xsl:element name="metadata">
  <xsl:element name="cause">
   <xsl:element name="changeID">
     OID.Temporal </xsl:element>
   <xsl:element name="changeCategory">
     <xsl:element name="changeType">
       Temporal Set Update </xsl:element>
     <xsl:element name="changeSchema">
       none </xsl:element>
   </xsl:element>
  </xsl:element>
  <xsl:element name="mappingInfo">
   <xsl:element name="mappingType">
     Absolute Direct </xsl:element>
   <xsl:element name="mappingMethod">
     Value </xsl:element>
   <xsl:element name="mappingProcess">
     Semi-automatical </xsl:element>
  </xsl:element>
  <xsl:element name="temporalHandlers">
   <xsl:element name="tempQueryHandle">
    <xsl:element name="entityChange">
     post_office </xsl:element>
   </xsl:element>
  </xsl:element>
 </xsl:element>
 <xsl:element name="audit"> not present </xsl:element>
</xsl:element>
</xsl:template>
```

The database storage of all objects and metadata can be done in relations tables including time stamp attributes for ensuring the temporal dimension of the data content, for example [25] have relation tables for a bi-temporal database. Alternatively in this example some data may be stored as XML, e.g. in an object-relational database or a native XML-database. However the pure relational table solution is more general, and can ensure a normalized database.

5 Conclusion and Future Research

5.1 Future Research

Future work includes considering if semantically knowledge from (temporal) sub-ontologies can increase the efficiency of generating new mappings between object versions. It is a question how to generate all needed mappings and their metadata in a cost-efficient semi-automatically way. Manually mapping will be very time-consuming and also over-complex, since the end user has to consider temporal changes that may have occurred in the past. A DRL system tool has to help the end user to firstly; find all the needed mappings, secondly; suggest possible mapping values for each mapping, and thirdly; help generating desirable related metadata. Such a mapping generation process can for example be related to an ontology evolution strategy described in [26].
Future research is needed to test how metadata can improve the query processing in the DRL-system in the respect of the query result quality. One question is how to raise the quality of inexact results from approximate mappings by active use of stored metadata about mappings and temporal changes in the DRL. Examples of implementation can be done in a conventional programming language, for example Java.

5.2 Conclusion

From a database point of view, the major problem for realization of our temporal data object framework is the handling of all kinds of mappings between versions of data content, schemas and possible ontologies. This paper provides some guidelines of adding interesting metadata elements to sub-mappings in our framework's object model. Such metadata include information about why the mapping is needed (the cause of data changes, in some cases this is trigged by temporal changes) and how the mapping is (was) generated. The metadata about a mapping between object versions can help in interpretation of data. The paper shows examples on such usage and relates these to the temporal perspectives in our case. We link the problems to the realization of a future integrated Electronic Health Record (EHR). However, it is in the nature of our framework that the usage is not limited to a particular application domain. The framework is general enough to represent and store any data content related to temporal perspectives.

References

1. Dick R.S, Steen E.B, Detmer D.E (Eds.). The Computer-Based Patient Record - An Essential Technology for Health Care, Revised ed., Institute Of Medicine, National Academy Press 1997.

2. Fagan L.M, Shortliffe E.H, The Future of Computer Applications in Health Care, Chapter 20 in Medical Informatics - Computer Applications In Health Care and Biomedicine, Springer-Verlag 2001, ISBN 0-387-98472-0
3. Office of Health and the Information Highway Health Canada. Canada Health Infoway: “Paths to Better Health”, Final Report of the Advisory Council on Health Infrastructure, February 1999 (http://www.hs-sc.gc.ca).
4. Waegemann C.P. The five levels of electronic health records, in M.D. Computing, Vol.13 No.3 1996
5. van Bemmel JH, Musen MA (ed.). Handbook of Medical Informatics, Springer 1997, ISBN 3-540-63351-0
6. Mallaug T, Bratbergsengen K. Long-term Temporal Data Representation of Personal Health Data, in LNCS 3631 / 2005, ADBIS 2005, Tallinn, Estonia, September 2005, Proceedings.
7. Roddick J F. A survey of schema versioning issues for database systems. Information and Software Technology, Vol. 37, No. 7, 1995, pp. 383-393.
8. http://www.w3.org/TR/xslt
9. Flouris G, Plexousakis D, Antoniou G. Evolving Ontology Evolution. SOFSEM 2006, Proceedings, LNCS 3831 Springer 2006.
10. Jensen et al. A consensus glossery of temporal database concepts. SIGMOD record, 23(1), 1994.
11. Clifford, Dyreson, Isakowitz, Jensen, Snodgrass. On the Semantics of ”now” in Databases. ACM Transactions on Database Systems, Vol. 22, No.2, 1997.
12. Rahm E, Bernstein P.A. A survey of approaches to automatic schema matching. The VLDB Journal 10, 2001.
13. Tan L, Katayama T. Meta Operations for Type Management in Object-Oriented Databases, DOOD 1989.
14. Nørvåg, K. VAGABOND The Design and Analysis of a Temporal Object Database Management System, Dr. ing. thesis, Norwegian University of Science and Technology, ISBN 82-7984-097-4
15. Berner et al. Will the Wave Finally Break? A Brief View of the Adoption of Electronic Medical Records in the United States, Journal of the American Medical Informatics Association, Volume 12, Number 1, Jan / Feb 2005.
16. Grimson J. Delivering the electronic healthcare record for the 21st century, in International Journal of Medical Informatics 64 (2001).
17. Tsiknakis, Katehakis, Orphanoudakis. A health information infrastructure enabling secure access to the life-long multimedia electronic health record, CARS 2004 / International Congress Series 1268 (Elsevier 2004).
18. Katehakis, Sfakianakis, Tsiknakis, Orphanoudakis. An Infrastructure for Integrated Electronic Health Record Services: The Role of XML (Extensible Markup Language), The Journal of Medical Internet Research, Volume 3, 2001 (http://www.jmir.org).
19. MedXML Consortium. http://www.medxml.net/E_mml30/MMLV3Spec.pdf
20. ANSI/HL7 Standard Version 2.4 Application Protocol for Electronic Data Exchange in Healthcare, 2000
21. Oliver, Shahar, Musen, Shortliffe. Representation of change in controlled medical terminologies, AI in Medicine, 15(1), 1999.
22. Salzano G, Bourret C. Identification and Composition of Metadata for Cooperative Information Systems – Illustration on the health information systems, at FIS 2005, http://www.mdpi.org/fis2005/
23. Gomez-Perez A., Fernandez-Lopez M., Corcho O. Ontological Engineering, Springer, 2004.
24. Prakash N, Srivastava S. Engineering Schema Transformation Methods, EMSISE 2003.

25. Wei H-C, Elmasri R. Study and Comparison of Schema Versioning and Data Conversion Techniques for Bi-temporal Databases, Sixth International Workshop on Temporal Representation and Reasoning, Orlando, Florida 1999.
26. Stojanovic L, Maedche A, Motik B, Stojanovic N. User-Driven Ontology Evolution Management, EKAW 2002, LNCS Volume 2473, 2002.